全国共同利用施設
東京大学情報基盤センター
Information Technology Center, The University of Tokyo

東京大学
THE UNIVERSITY OF TOKYO

Communications in ppOpen-MATH/MG

Kengo Nakajima
Information Technology Center, The University of Tokyo, Japan

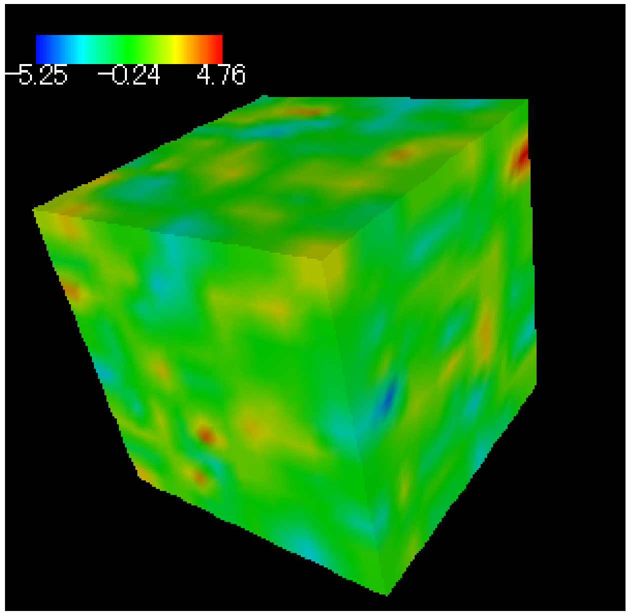
October 28th, 2013

Target Application: *pGW3D-FVM*

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson's equation
  - Randomly distributed water conductivity

$$\nabla \cdot \left(\lambda(x, y, z) \nabla \phi \right) = q, \phi = 0 \ at \ z = z_{\max}$$

  - Distribution of water conductivity is defined through methods in geostatistics 〔Deutsch & Journel, 1998〕
- Finite-Volume Method on Cubic Voxel Mesh
- Distribution of Water Conductivity
  - 10^{-5}-10^{+5}, Condition Number ~ 10^{+10}
  - Average: 1.0
- Cyclic Distribution: 128^3

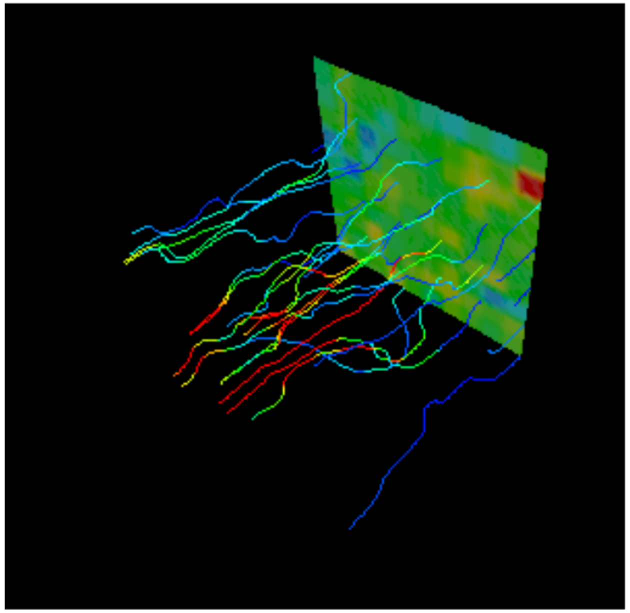


Target Application: *pGW3D-FVM*

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson's equation
  - Randomly distributed water conductivity

$$\nabla \cdot \left(\lambda(x, y, z) \nabla \phi \right) = q, \ \phi = 0 \ at \ z = z_{\max}$$

  - Distribution of water conductivity is defined through methods in geostatistics〔Deutsch & Journel, 1998〕
- Finite-Volume Method on Cubic Voxel Mesh



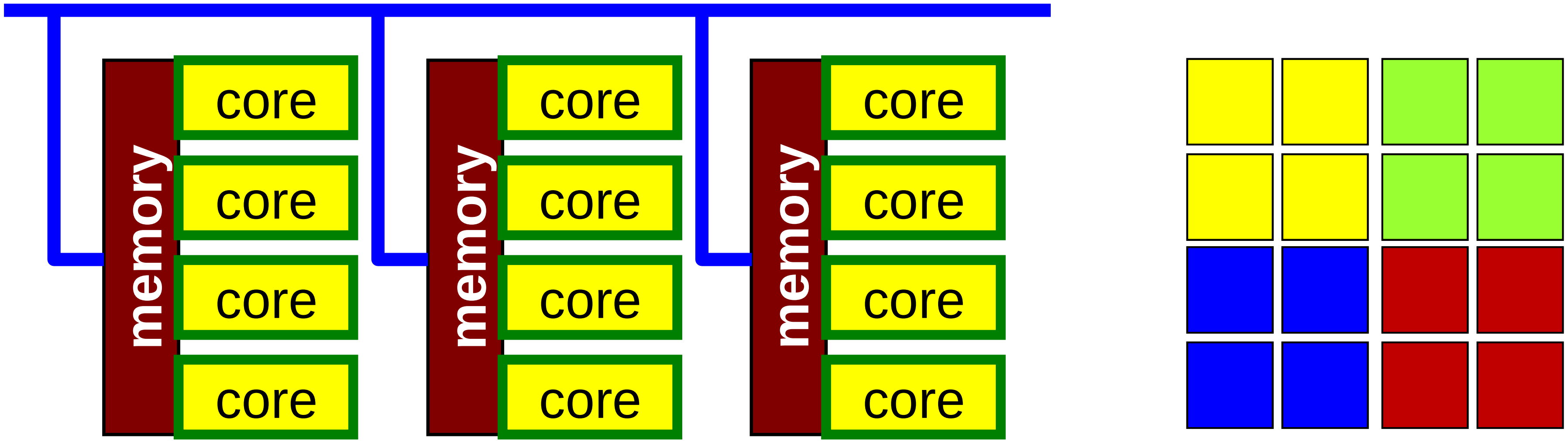
- Distribution of Water Conductivity
  - 10^{-5}-10^{+5}, Condition Number ~ 10^{+10}
  - Average: 1.0
- Cyclic Distribution: 128^3

Motivation of This Study

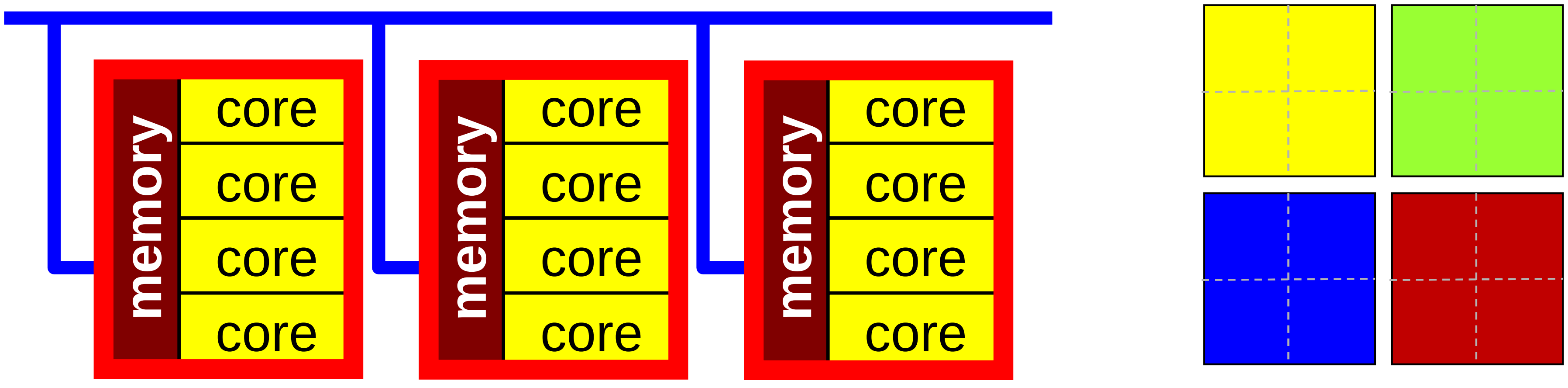
- Large-scale 3D Groundwater Flow
  - Poisson equations
  - Heterogeneous porous media
- Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)
- Flat MPI vs. Hybrid (OpenMP+MPI)
- Expectations for Hybrid Parallel Programming Model
  - Number of MPI processes (and sub-domains) to be reduced
  - O(10^8-10^9)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

Flat MPI vs. Hybrid

Flat-MPI：Each Core -> Independent

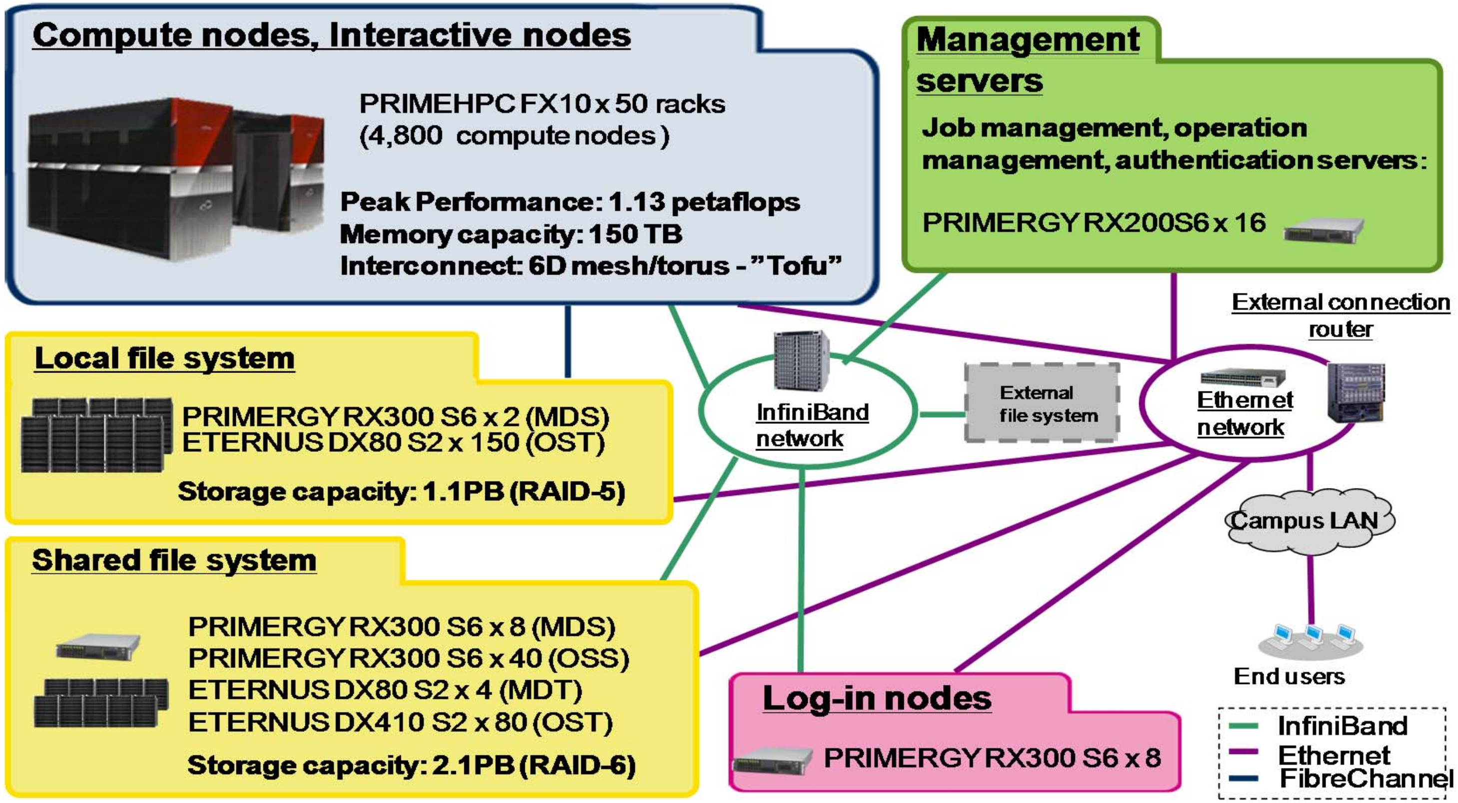


Hybrid：Hierarchal Structure



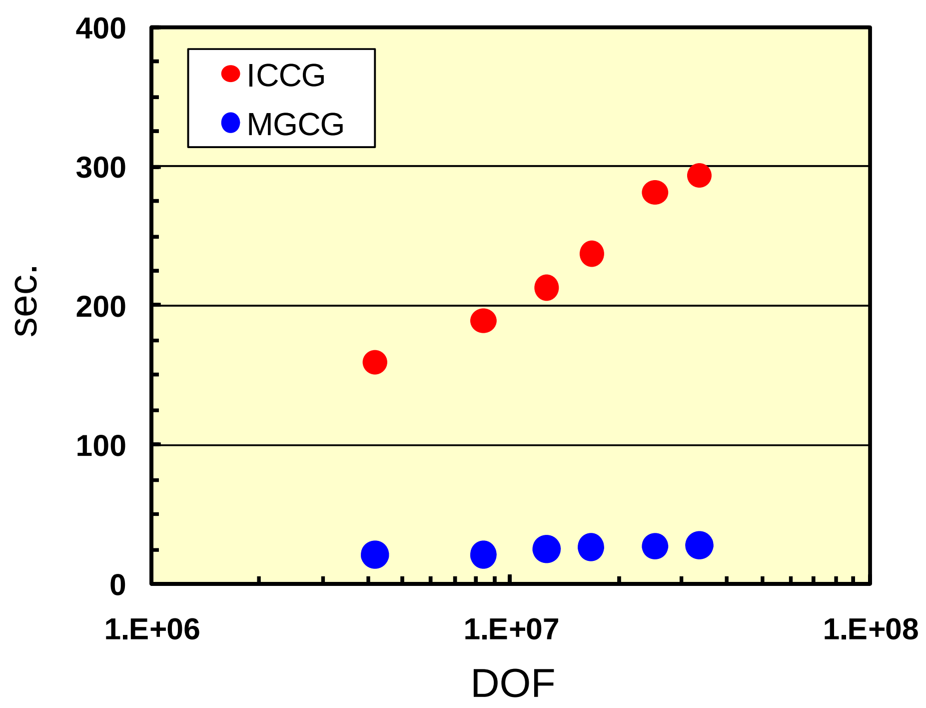
Fujitsu PRIMEHPC FX10 (Oakleaf-FX) at the U. Tokyo

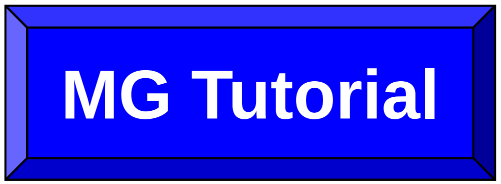
- SPARC64 Ixfx (4,800 nodes, 76,800 cores)
- Commercial version of K computerx
- Peak: 1.13 PFLOPS (1.043 PF, 21st, 40th TOP 500 in 2012 Nov.)
- Memory BWTH 398 TB/sec.



Multigrid

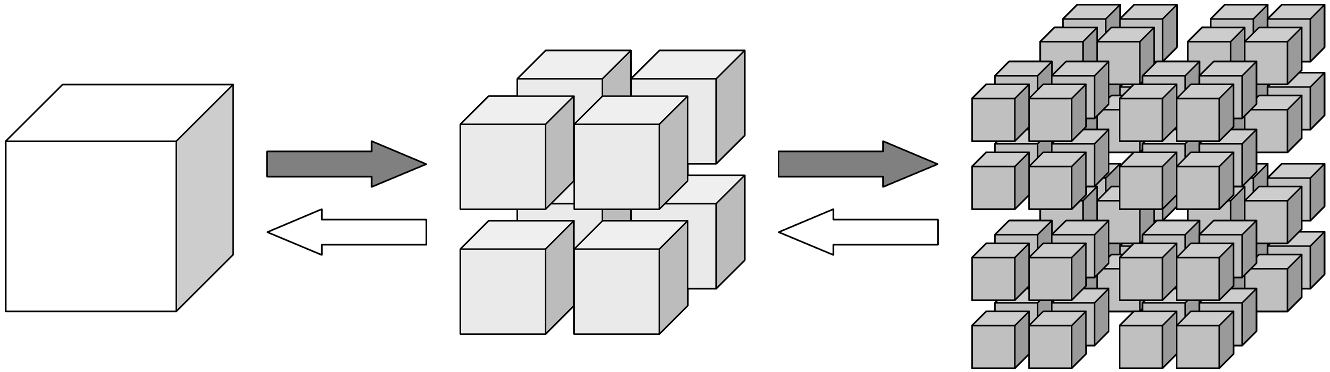
- Scalable Multi-Level Method using Multilevel Grid for Solving Linear Eqn's
  - Computation Time ~ O(N) (N: # unknowns)
  - Good for large-scale problems
- Preconditioner for Krylov Iterative Linear Solvers
  - MGCG





Linear Solvers

- Preconditioned CG Method
  - Multigrid Preconditioning (MGCG)
  - IC(0) for Smoothing Operator (Smoother): good for ill-conditioned problems
- Parallel Geometric Multigrid Method
  - 8 fine meshes (children) form 1 coarse mesh (parent) in isotropic manner (octree)
  - V-cycle
  - Domain-Decomposition-based: Localized Block-Jacobi, Overlapped Additive Schwartz Domain Decomposition (ASDD)
  - Operations using a single core at the coarsest level (redundant)

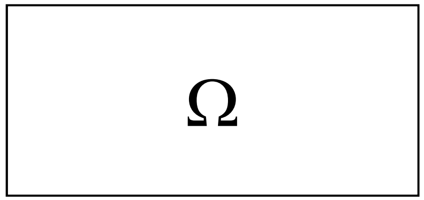


Overlapped Additive Schwartz Domain Decomposition Method

ASDD: Localized Block-Jacobi Precond. is stabilized

<u>Global Operation</u>

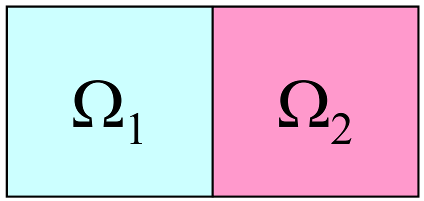
$$Mz = r$$



<u>Local Operation</u>

$$z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2}$$

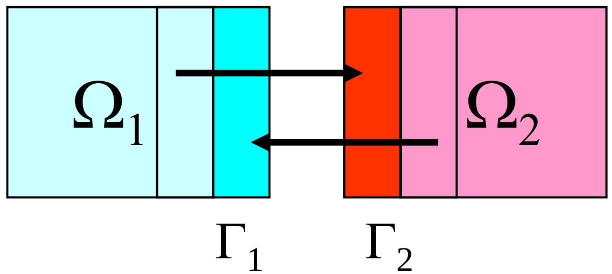
Ω_i : Internal（i≦N）
Γ_i : External（i＞N）



<u>Global Nesting Correction</u>

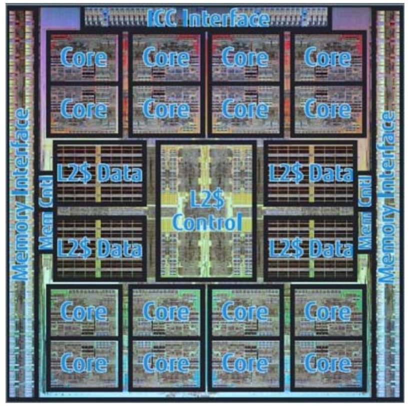
$$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}(r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1})$$

$$z_{\Omega_2}^{n} = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1}(r_{\Omega_2} - M_{\Omega_2} z_{\Omega_2}^{n-1} - M_{\Gamma_2} z_{\Gamma_2}^{n-1})$$

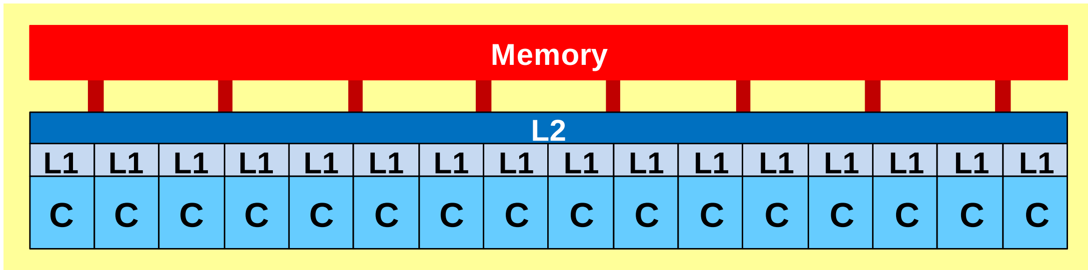


Computations on Fujitsu FX10

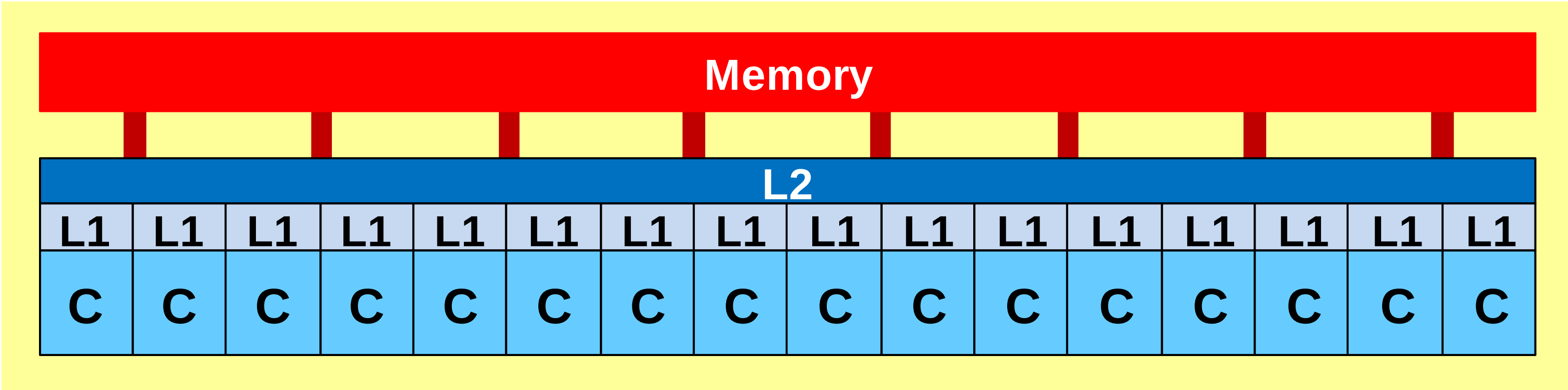
- Fujitsu PRIMEHPC FX10 at U.Tokyo (Oakleaf-FX)
  - 16 cores/node, flat/uniform access to memory
- Up to 4,096 nodes (65,536 cores) (Large-Scale HPC Challenge)
  - Max 17,179,869,184 unknowns
  - Flat MPI, HB 4x4, HB 8x2, HB 16x1
    - HB MxN: M-threads x N-MPI-processes on each node



- Weak Scaling
  - 64^3 cells/core
- Strong Scaling
  - $128^3 \times 8$= 16,777,216 unknowns, from 8 to 4,096 nodes



- Network Topology is not specified
  - 1D



HB M x N

Number of OpenMP threads per a single MPI process

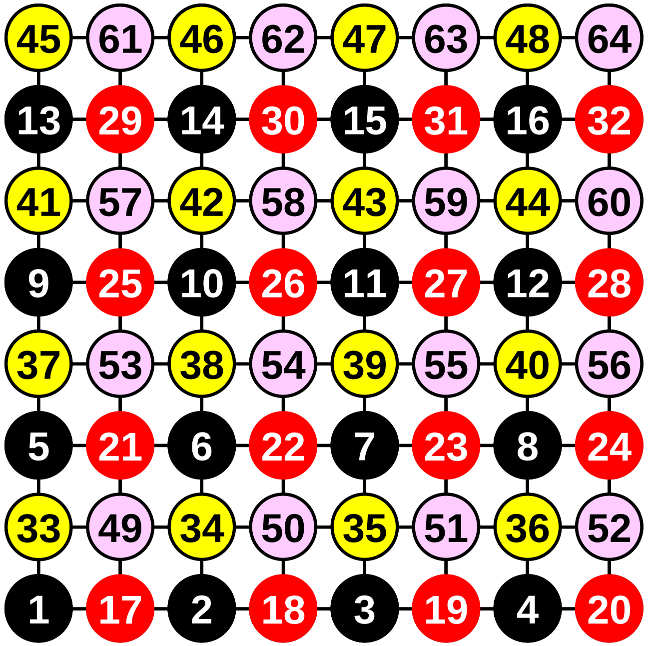
Number of MPI process per a single node

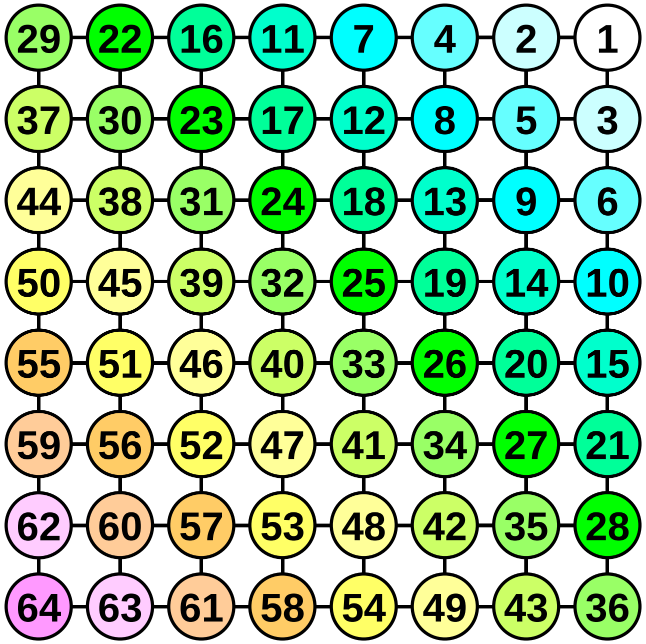
Reordering for extracting parallelism in each domain (= MPI Process)

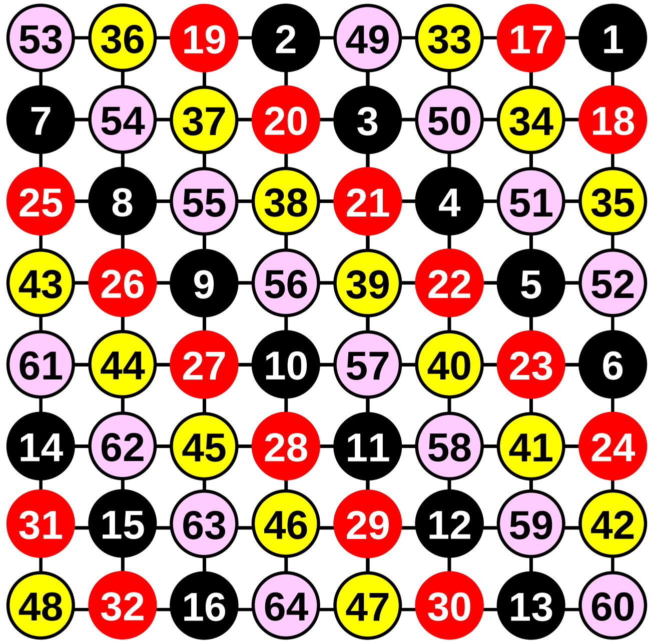
- Krylov Iterative Solvers
  - Dot Products
  - SMVP
  - DAXPY
  - Preconditioning
- IC/ILU Factorization, Forward/Backward Substitution
  - Global Dependency
  - Reordering needed for parallelism ([KN 2003] on the Earth Simulator, KN@CMCIM-2002)
  - Multicoloring, RCM, CM-RCM

Ordering Methods

Elements in “same color” are independent: to be parallelized







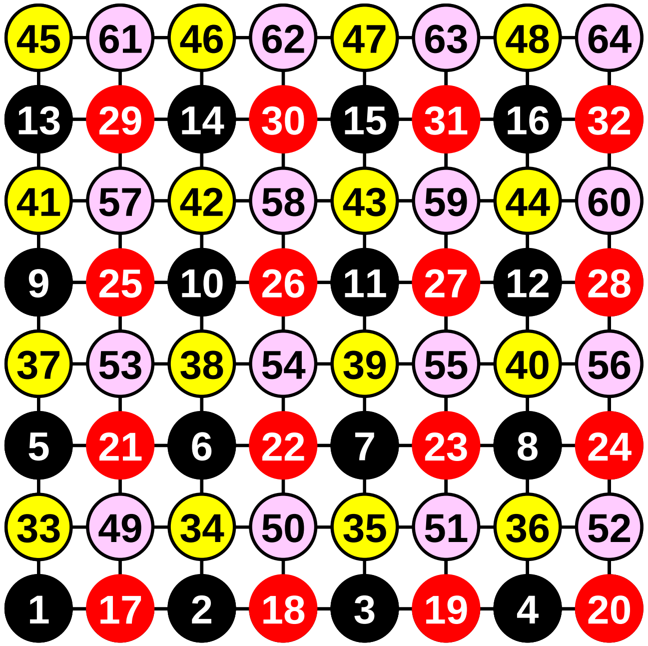
MC (Color#=4)
Multicoloring

RCM
Reverse Cuthill-Mckee

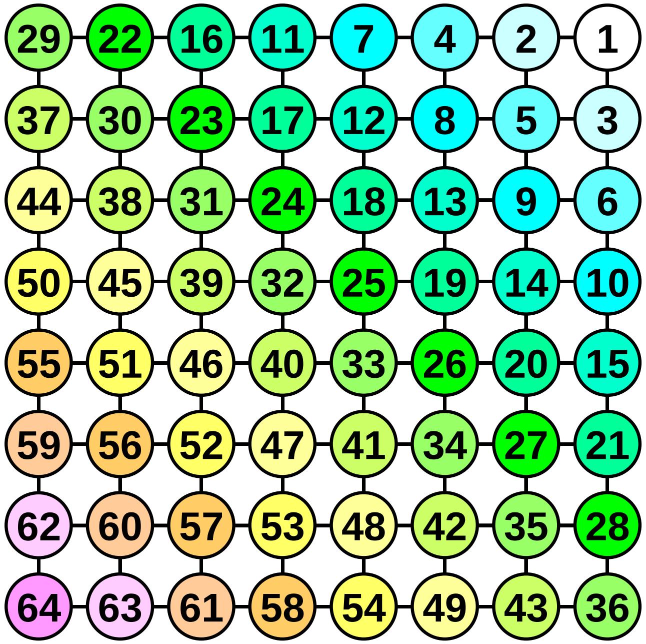
CM-RCM (Color#=4)
Cyclic MC + RCM

Ordering Methods

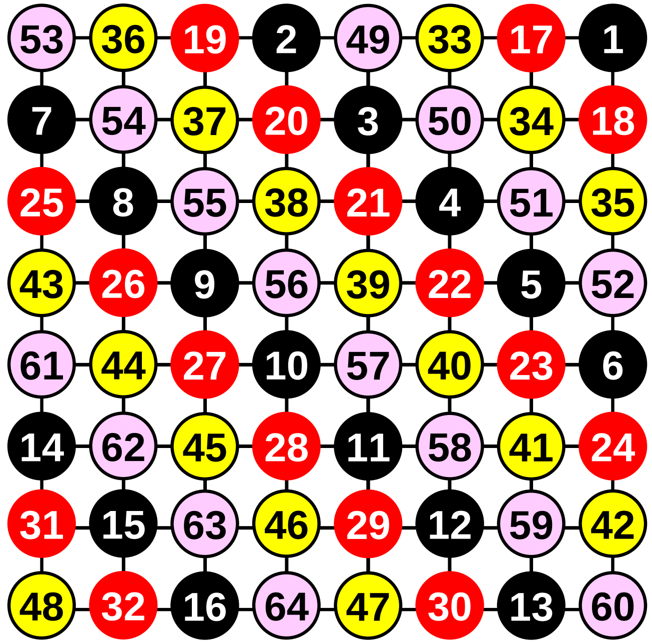
Elements in “same color” are independent: to be parallelized



MC (Color#=4)
Multicoloring



RCM
Reverse Cuthill-Mckee



CM-RCM (Color#=4)
Cyclic MC + RCM

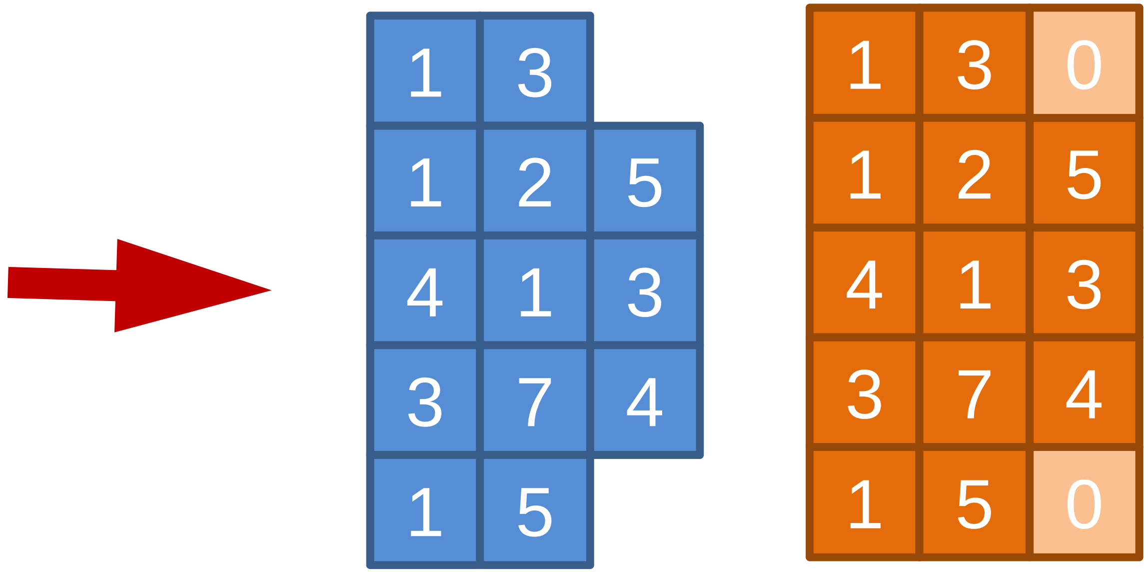
What is new in this work ?

- Storage format of coefficient matrices
  - CRS (Compressed Row Storage): Original
  - ELL (Ellpack-Itpack)

- Coarse Grid Aggregation (CGA)
- Hierarchical CGA: Communication Reducing CGA

ELL: Fixed Loop-length, Nice for Pre-fetching

$$\begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ 1 & 2 & 5 & 0 & 0 \\ 4 & 1 & 3 & 0 & 0 \\ 0 & 3 & 7 & 4 & 0 \\ 1 & 0 & 0 & 0 & 5 \end{bmatrix}$$

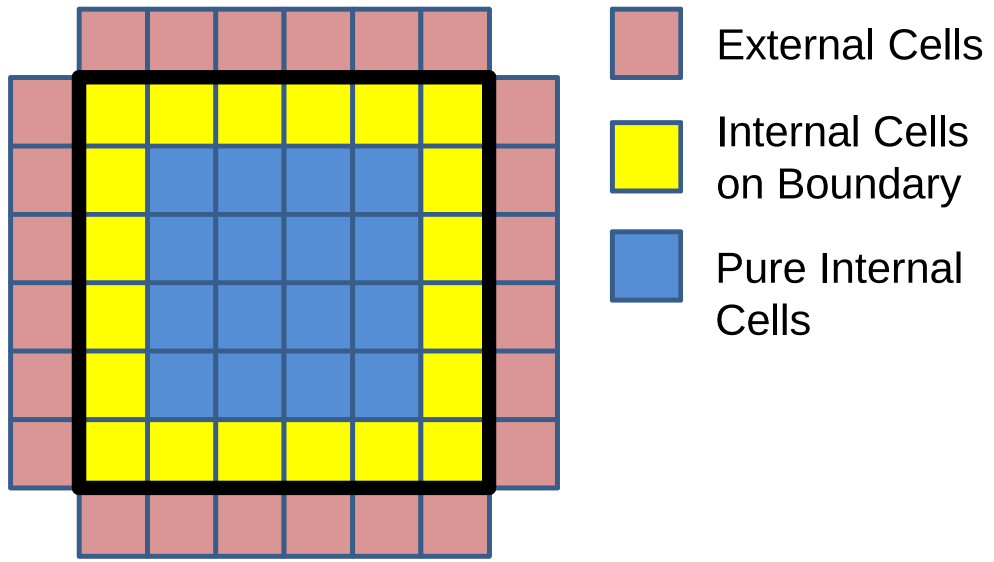


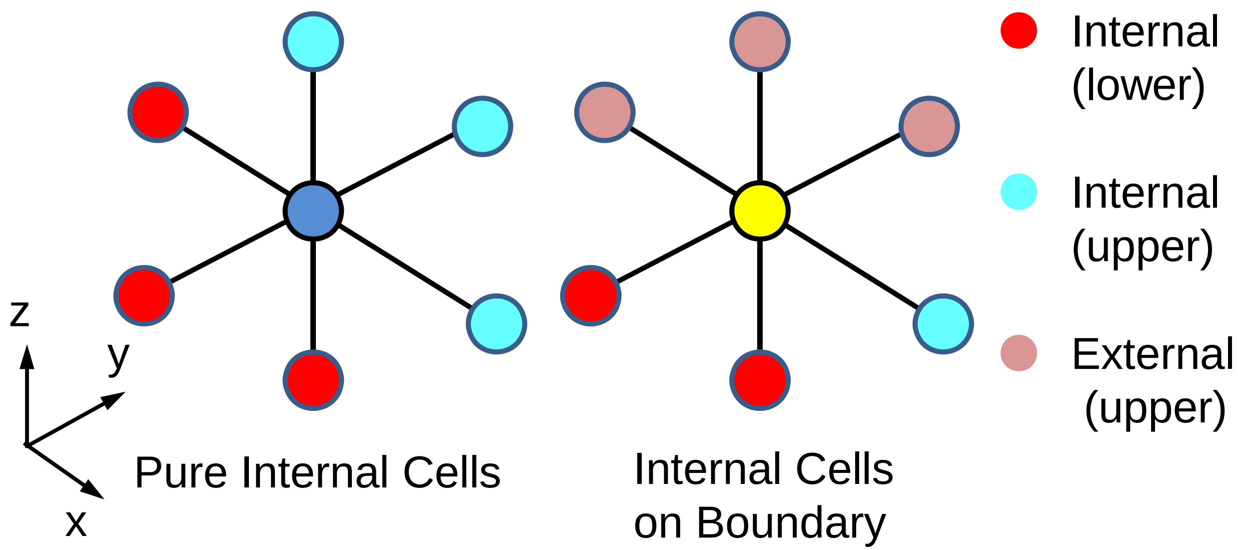
(a) CRS (b) ELL

Special Treatment for "Boundary" Cells connected to "Halo"

- Distribution of Lower/Upper Non-Zero Off-Diagonal Components

- Pure Internal Cells
  - L: ~3, U: ~3
- Boundary Cells
  - L: ~3, U: ~6





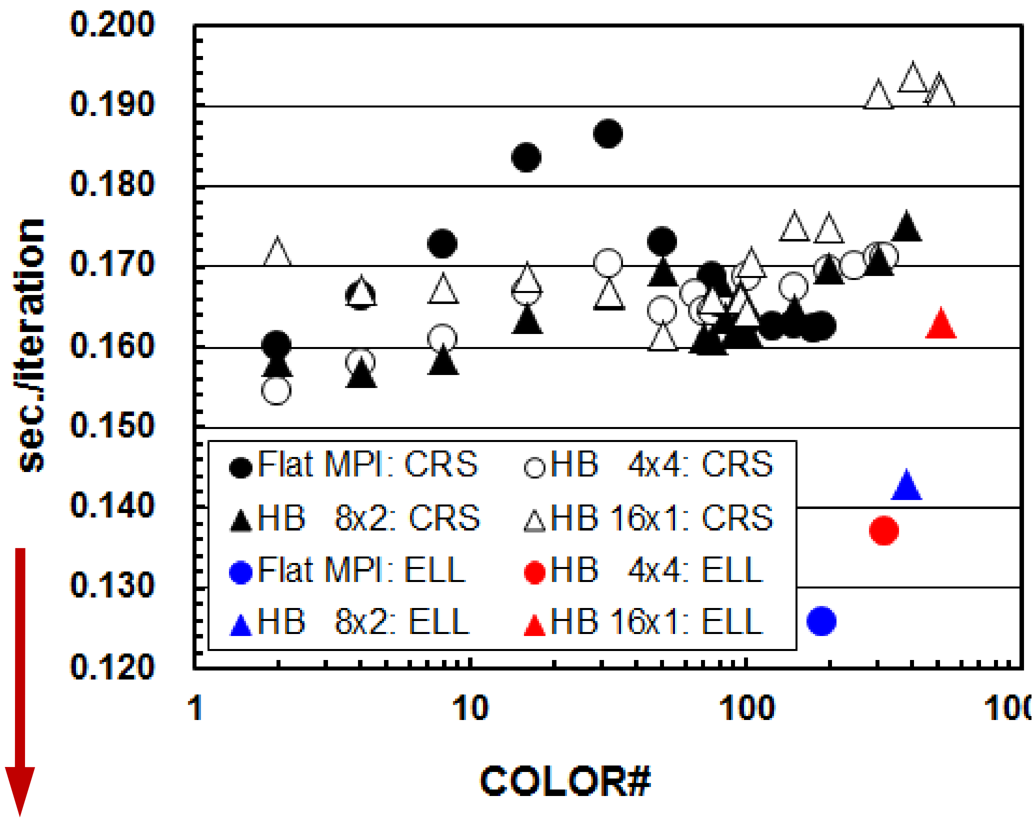
Effect of CRS/ELL

4 nodes, 64 cores, (16,777,216 meshes: 64^3 meshes/core)

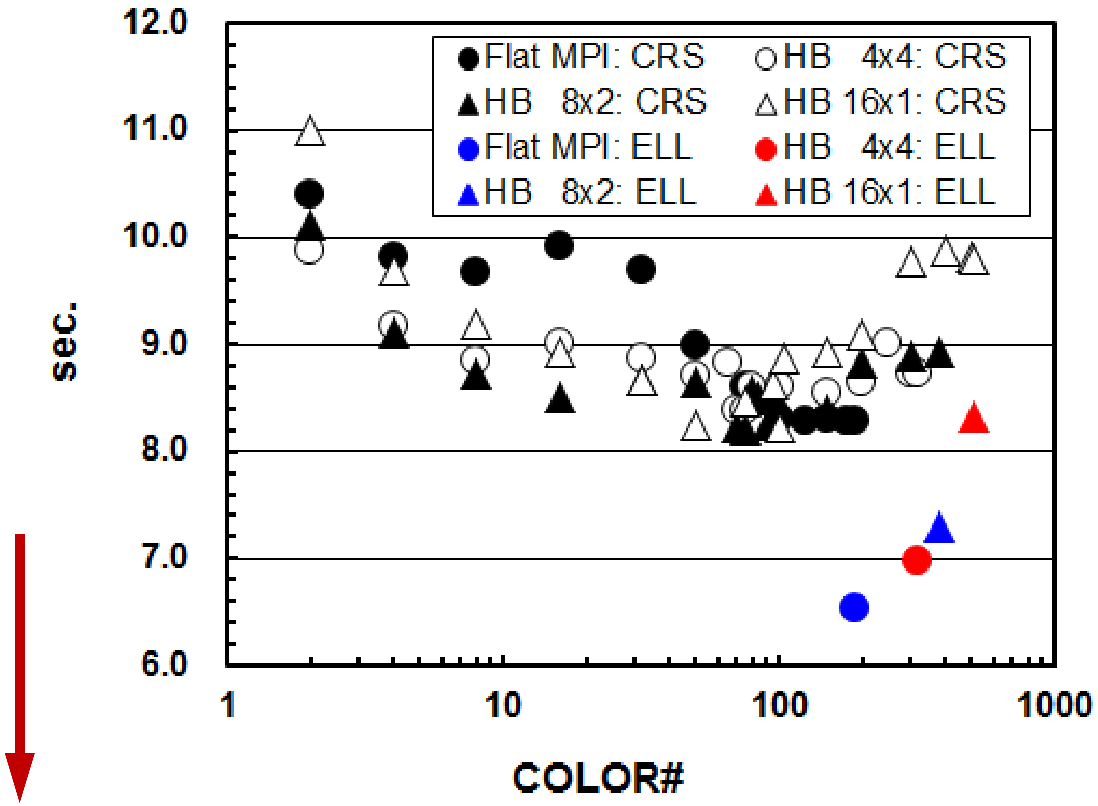
CM-RCM(k), only RCM for ELL cases

DOWN is GOOD

sec./iteration



time for MGCG



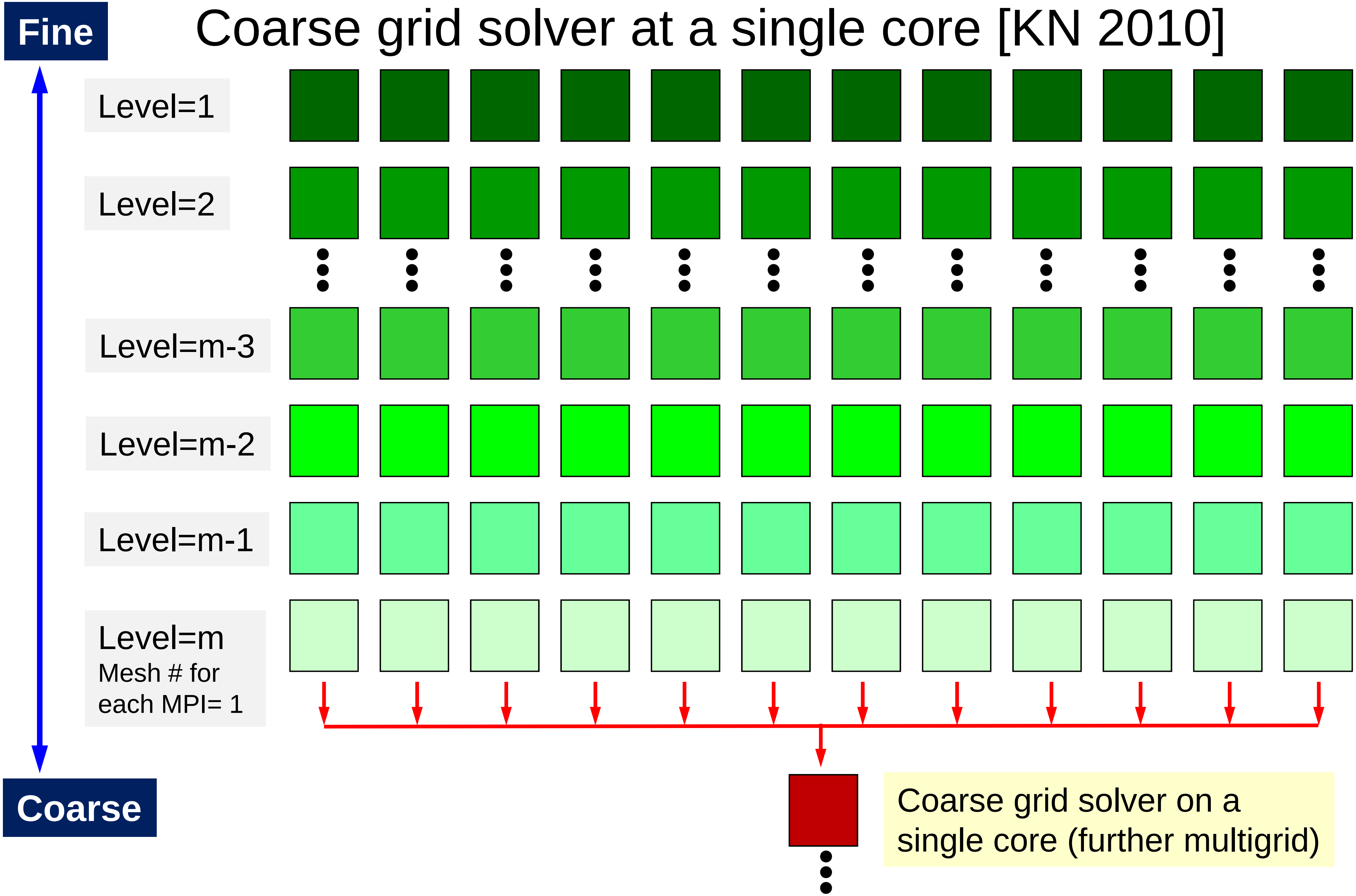
Down is good

Analyses by Detailed Profiler of Fujitsu FX10, single node, Flat MPI, RCM (Multigrid Part)

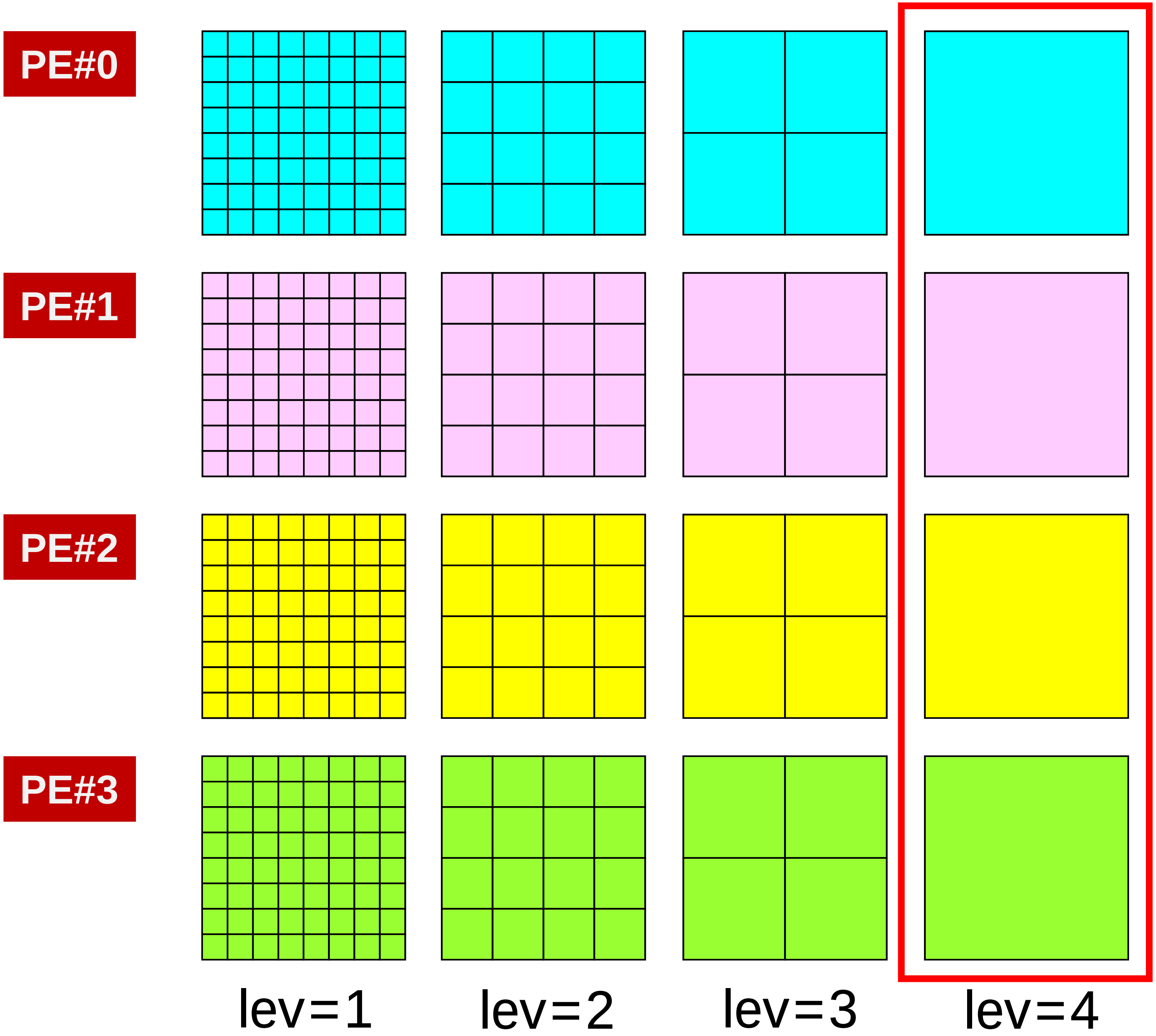
	L1-cache Demand Miss	Instructions	Time for Multigrid	Operation Wait
CRS	29.3%	1.447×10^{10}	6.815 sec.	1.453 sec.
ELL	16.5%	6.385×10^{9}	5.457 sec.	0.312 sec.

Original Approach (restriction)

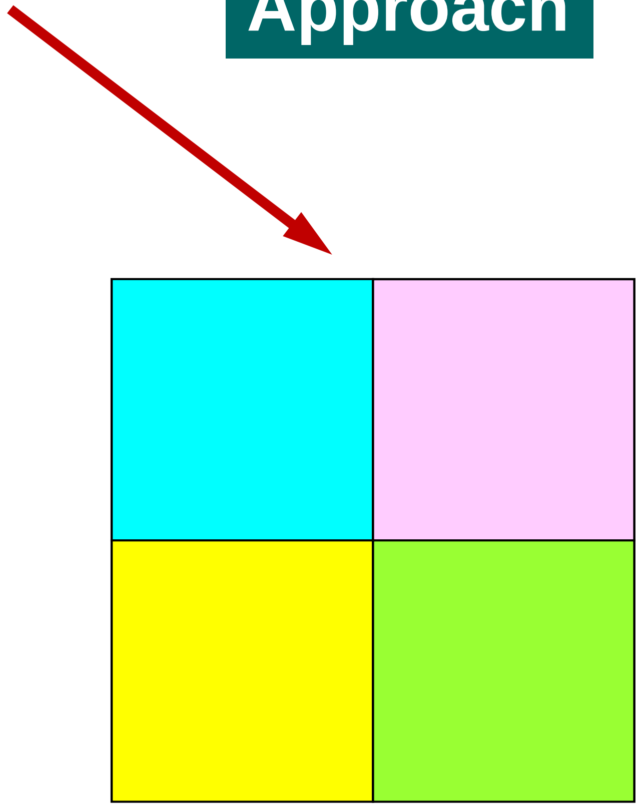
Coarse grid solver at a single core [KN 2010]



Coarse Grid Solver on a Single Core



Original Approach

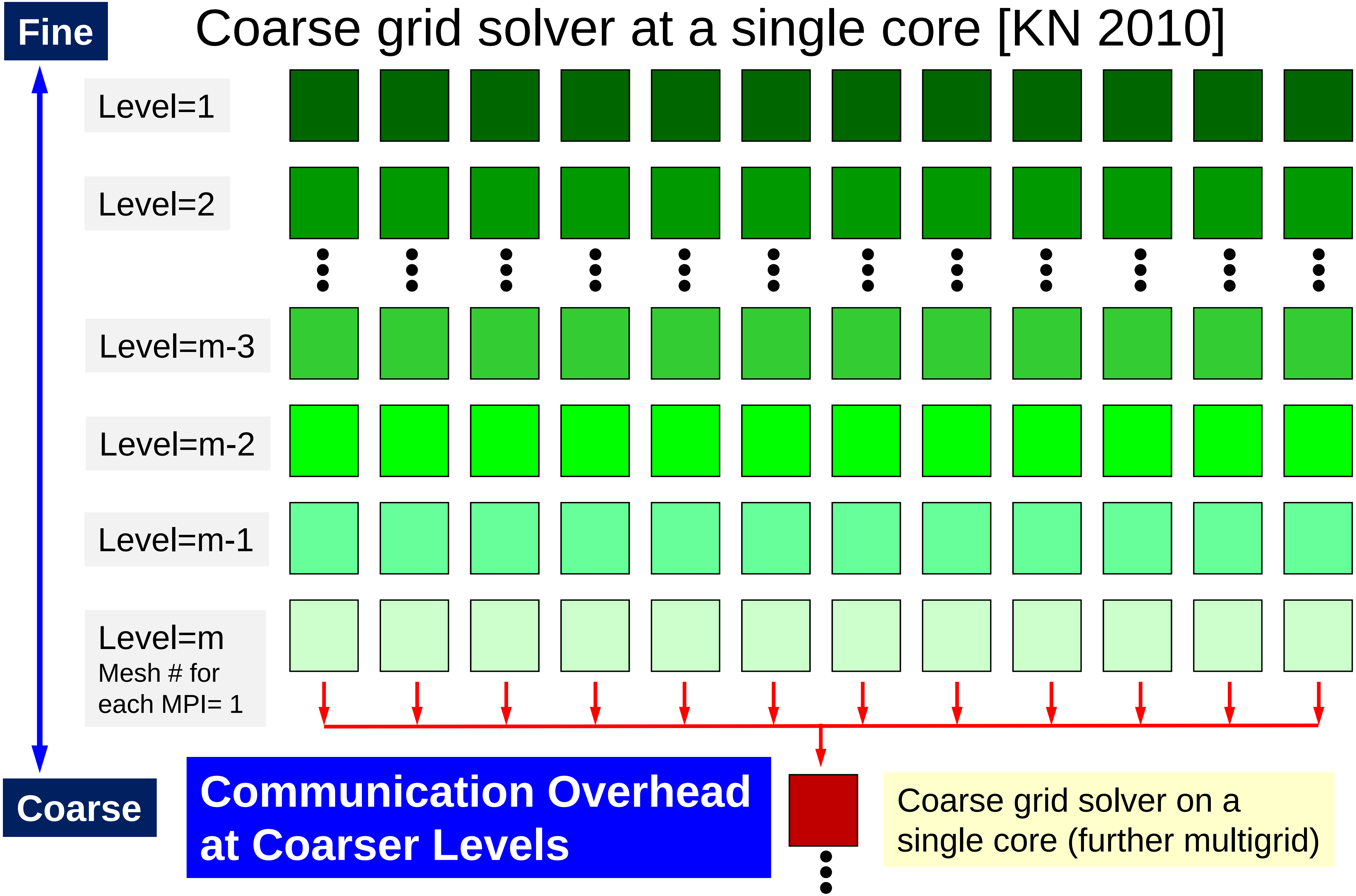


Size of the Coarsest Grid= Number of MPI Processes Redundant Process

In Flat-MPI, this size is larger

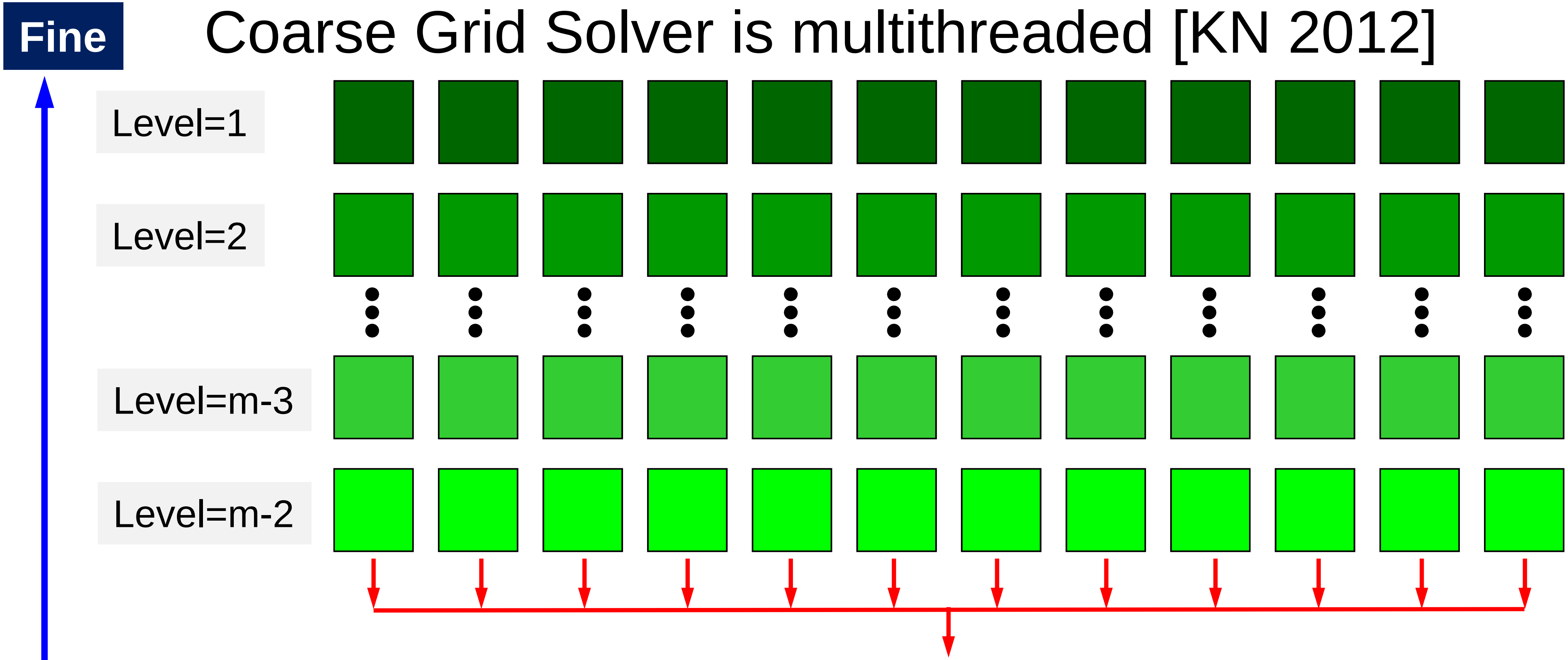
Original Approach (restriction)

Coarse grid solver at a single core [KN 2010]

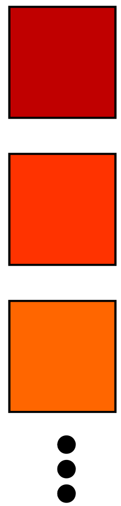


Coarse Grid Aggregation (CGA)

Coarse Grid Solver is multithreaded [KN 2012]



- Communication overhead could be reduced
- Coarse grid solver is more expensive than original approach.
- If process number is larger, this effect might be significant



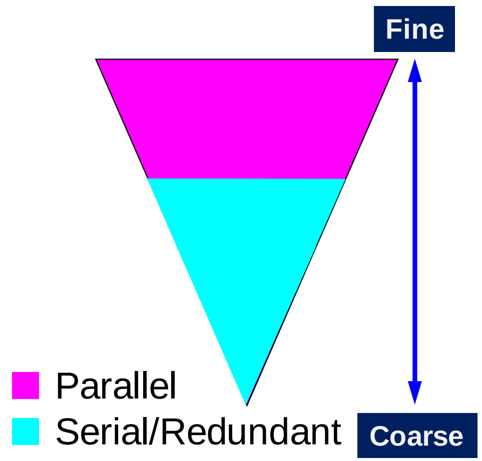
Coarse grid solver on a single MPI process (multi-threaded, further multigrid)

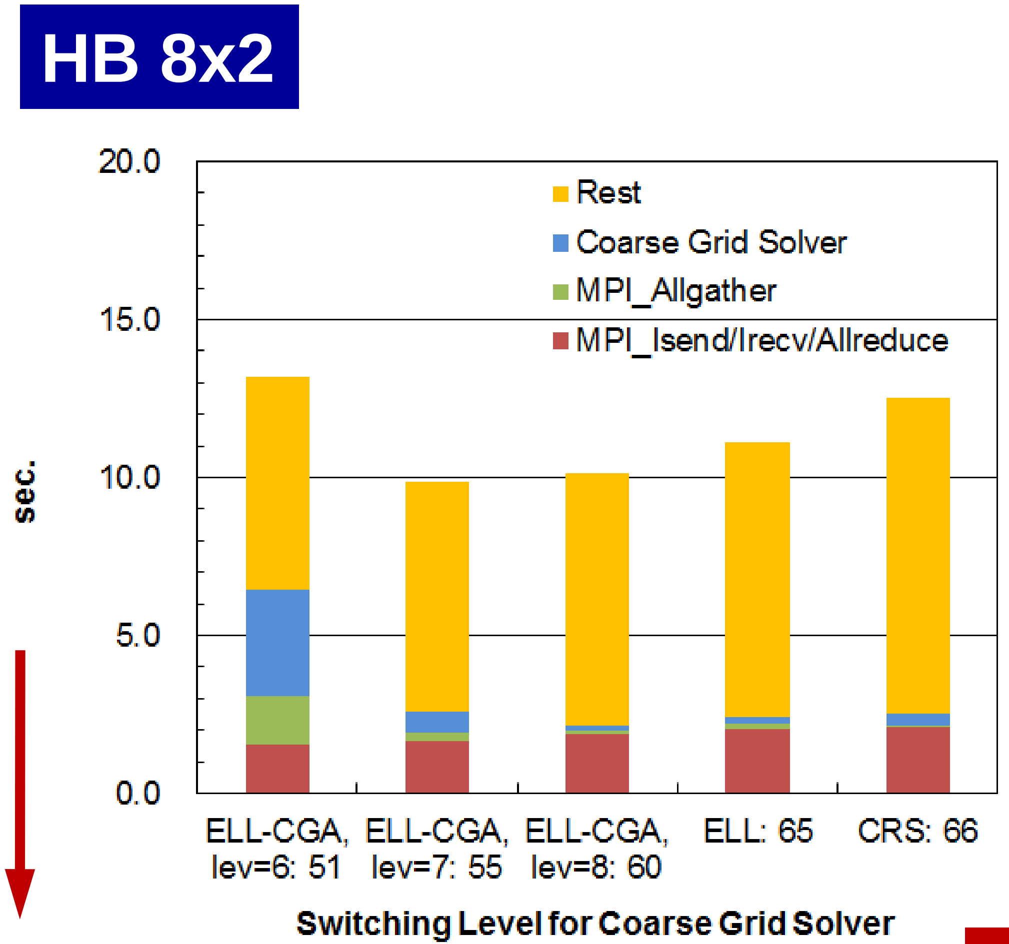
Results at 4,096 nodes

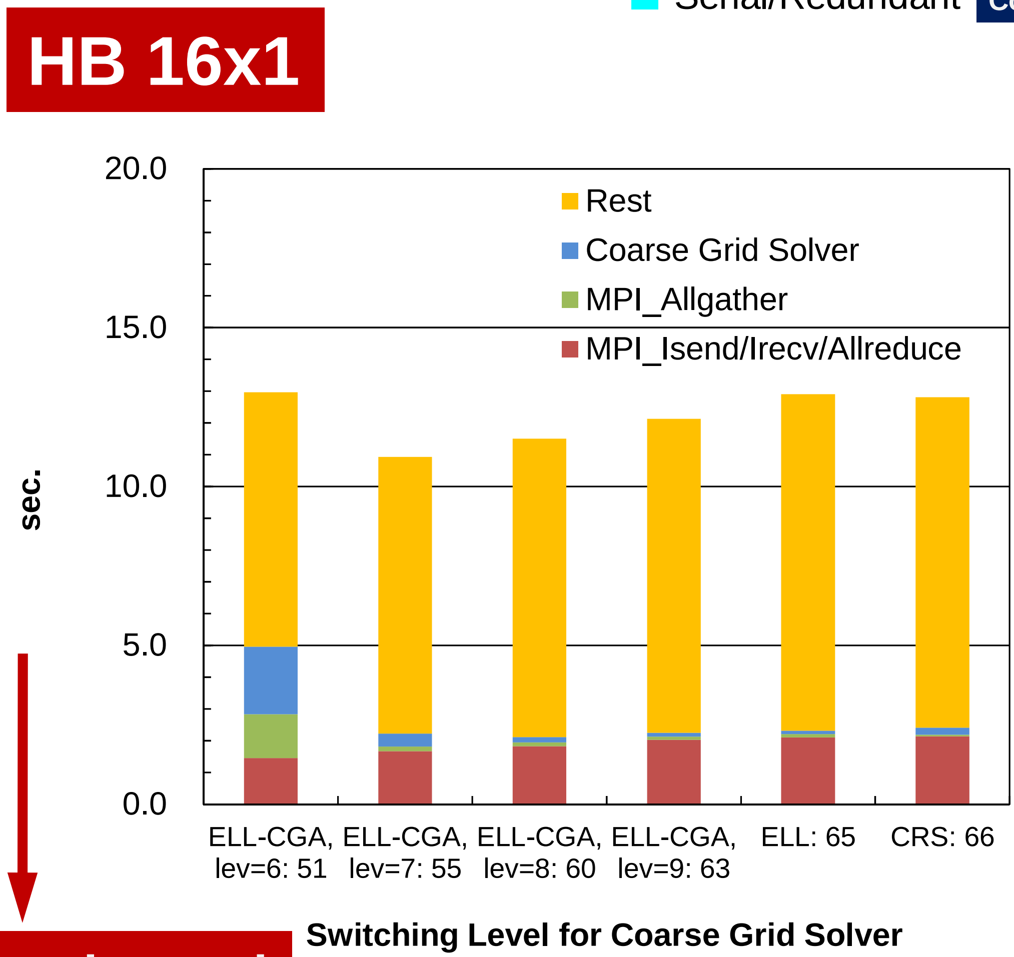
lev: switching level to "coarse grid solver"

Opt. Level= 7, HB 8x8 is the best

DOWN is GOOD





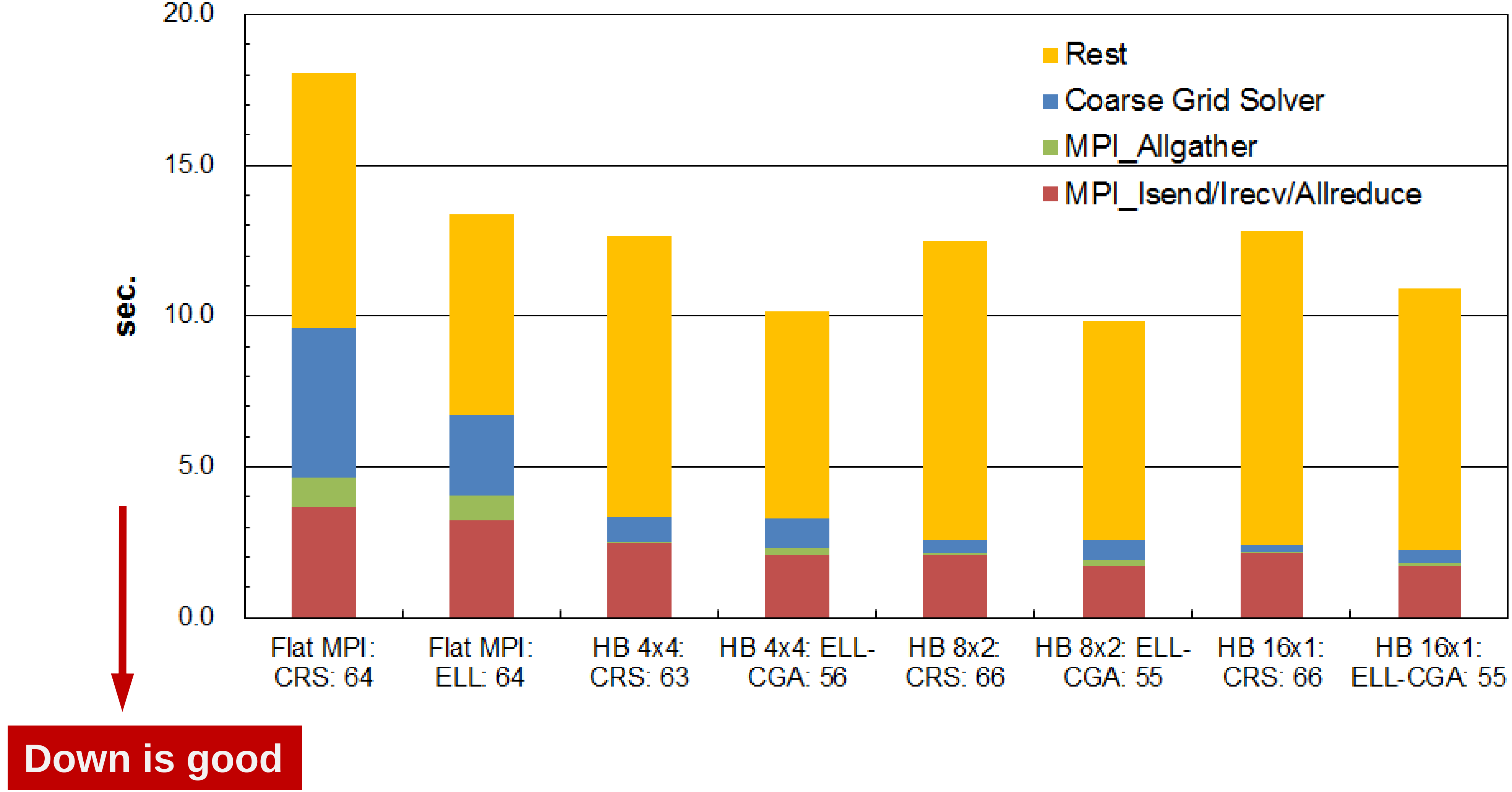


Down is good

Weak Scaling at 4,096 nodes

17,179,869,184 meshes (64^3 meshes/core)

best switching level (=7)

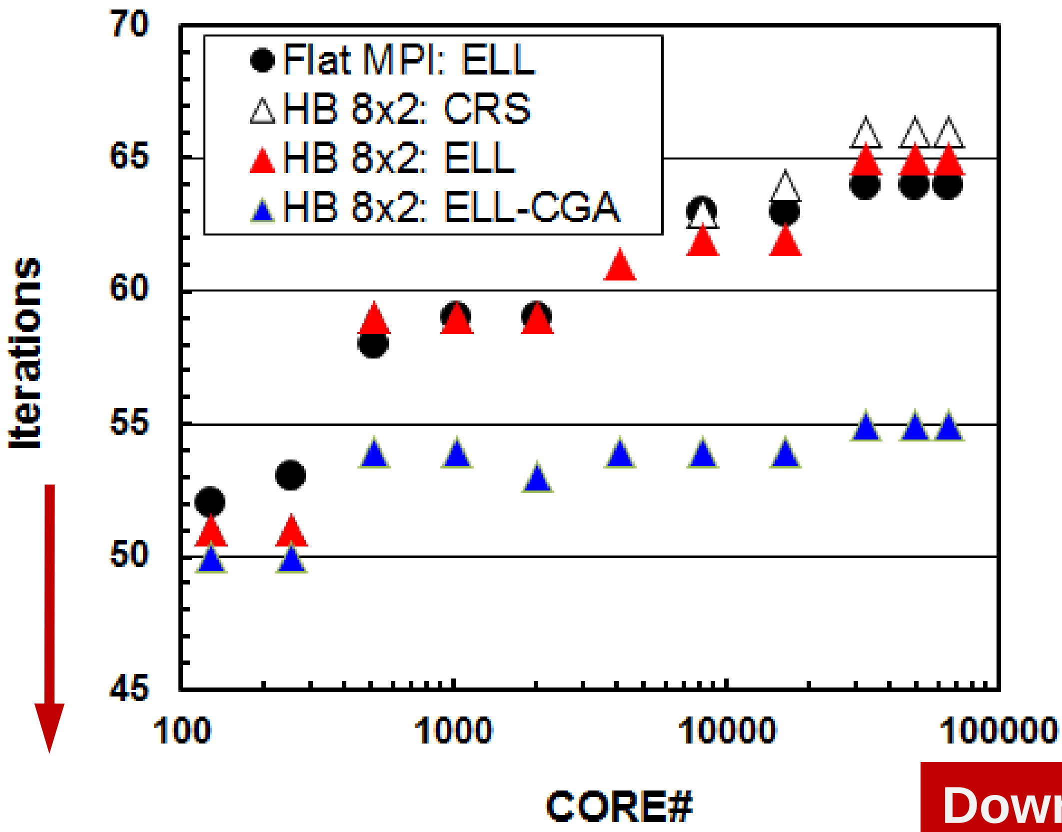


Weak Scaling: up to 4,096 nodes

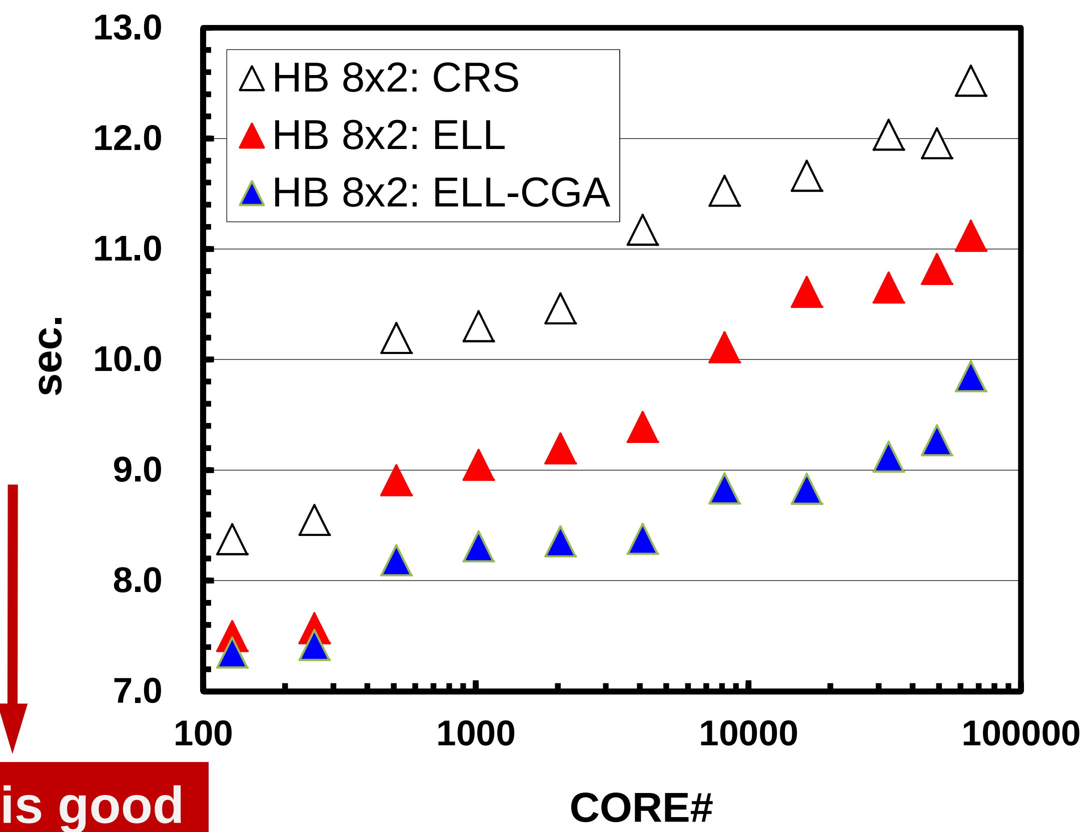
up to 17,179,869,184 meshes (64^3 meshes/core)

Convergence has been much improved by coarse grid aggregation, DOWN is GOOD

Iterations



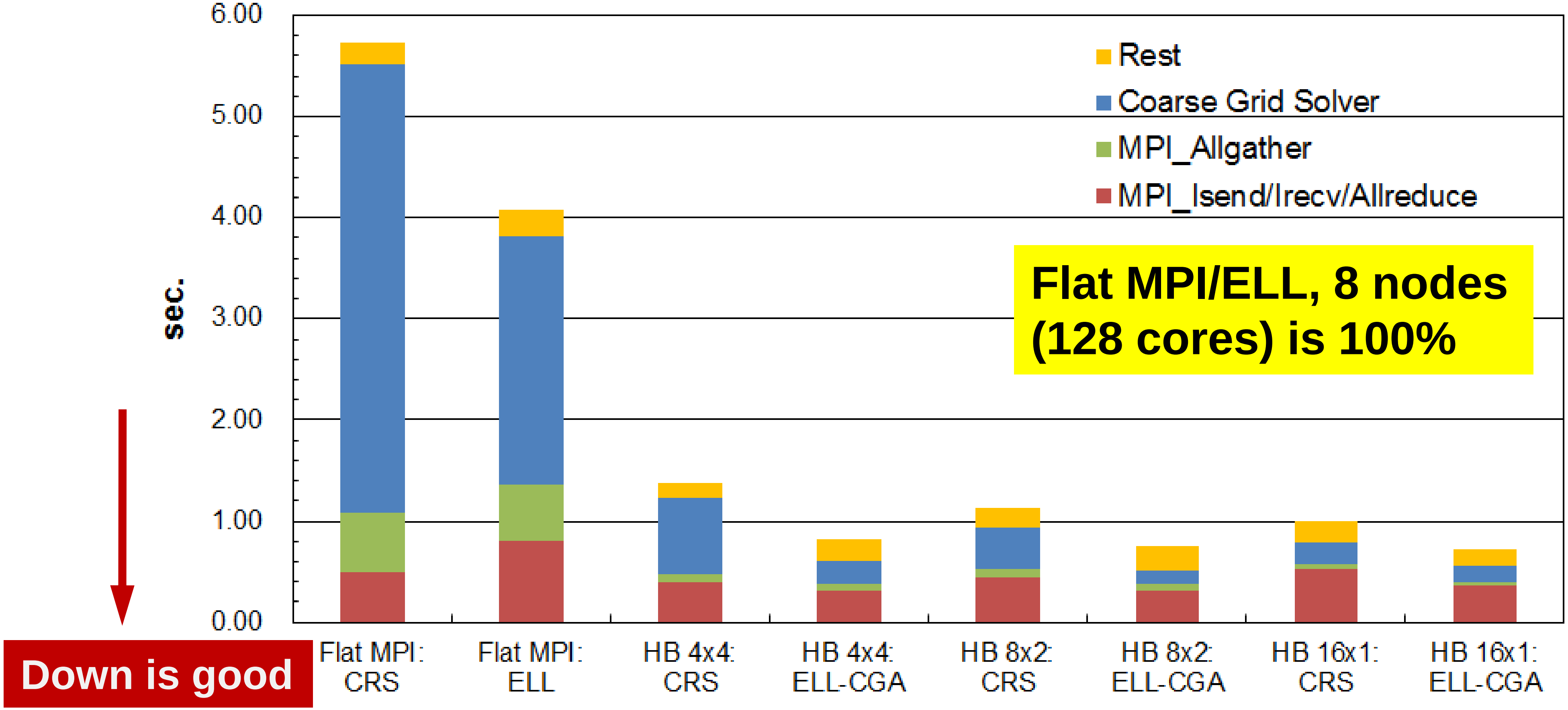
sec.



Down is good

Strong Scaling at 4,096 nodes

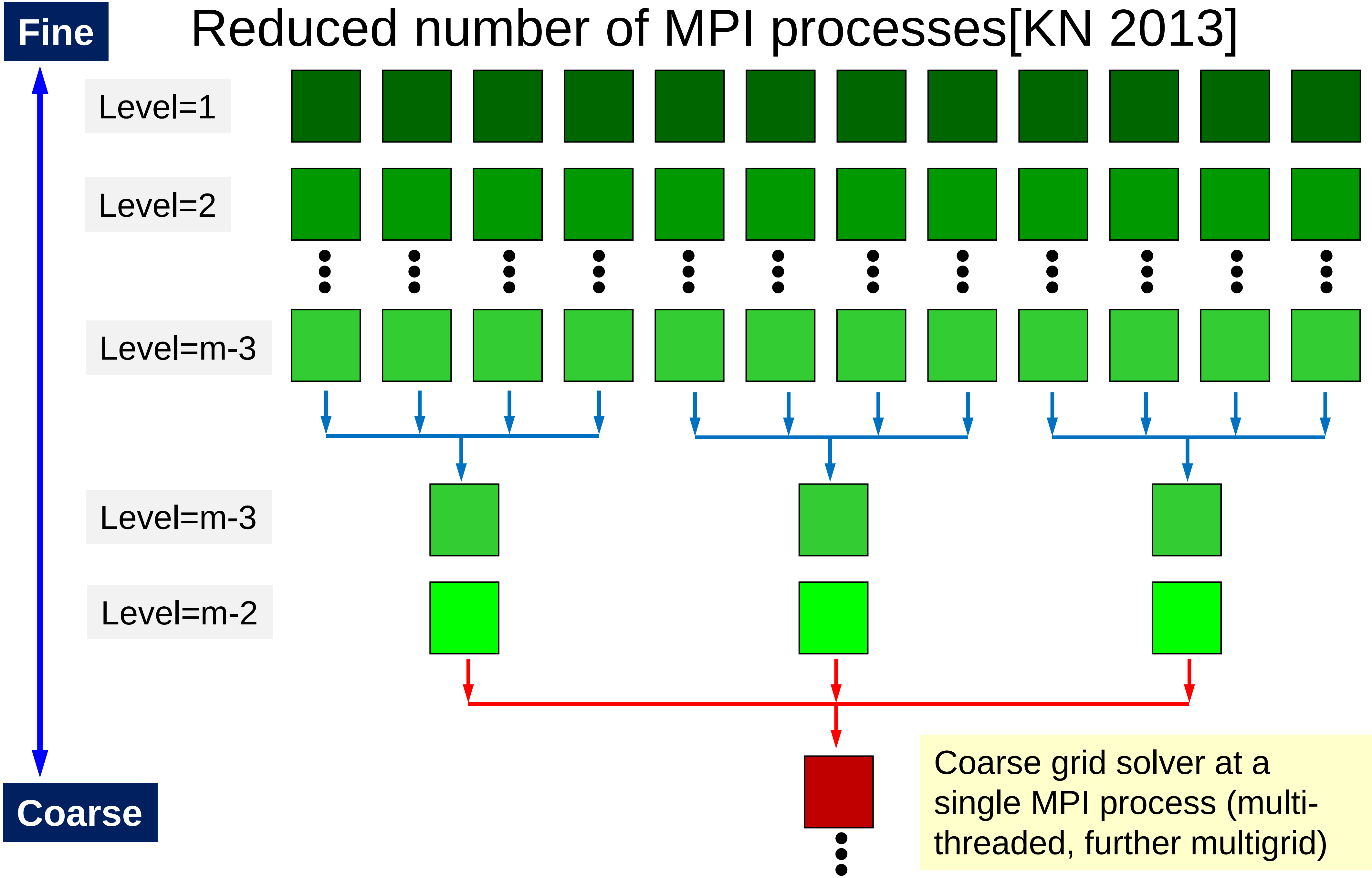
268,435,456 meshes, only 16^3 meshes/core at 4,096 nodes



	Flat MPI		HB 4×4		HB 8×2		HB 16×1	
	CRS	ELL	CRS	ELL-CGA	CRS	ELL-CGA	CRS	ELL-CGA
Iterations until Convergence	57	58	58	46	63	49	63	51
MGCG solver (sec.)	5.73	4.07	1.38	.816	1.13	.749	1.00	.714
Parallel performance (%)	2.02	2.85	8.38	14.2	10.3	15.5	11.6	16.2

Hierarchical CGA: Comm. Reducing MG

Reduced number of MPI processes[KN 2013]

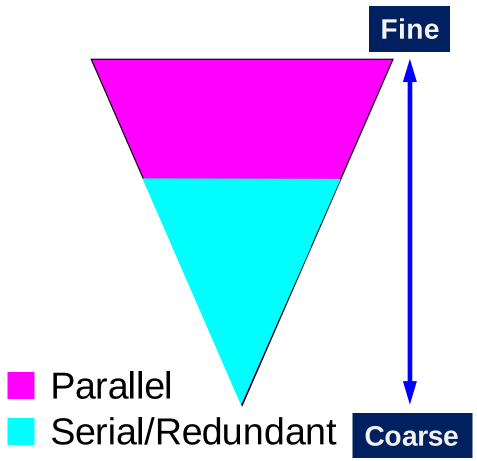


Results at 4,096 nodes

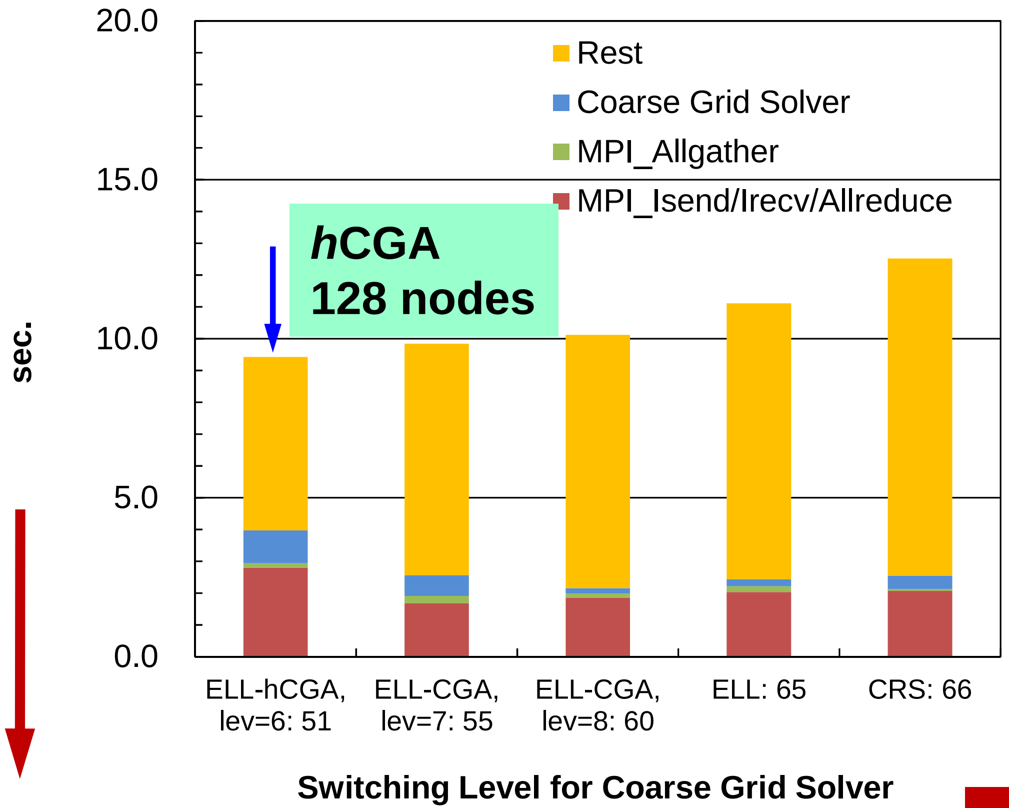
lev: switching level to “coarse grid solver”

Opt. Level= 7, HB 8x8 is the best

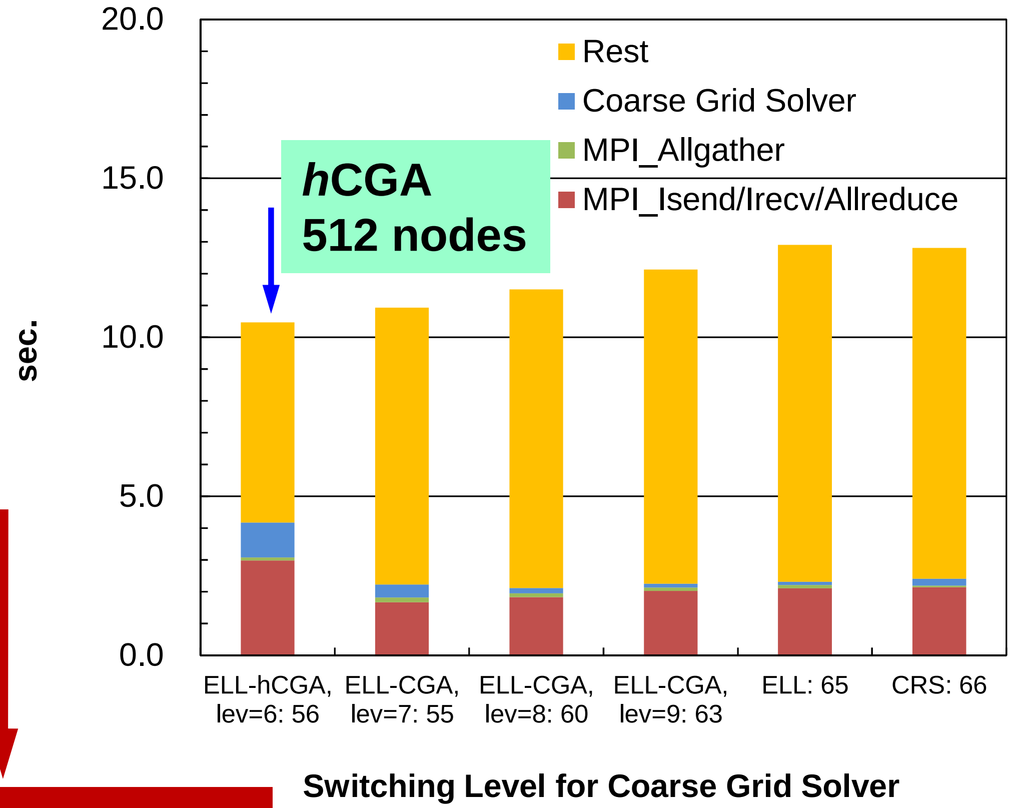
DOWN is GOOD



HB 8x2



HB 16x1



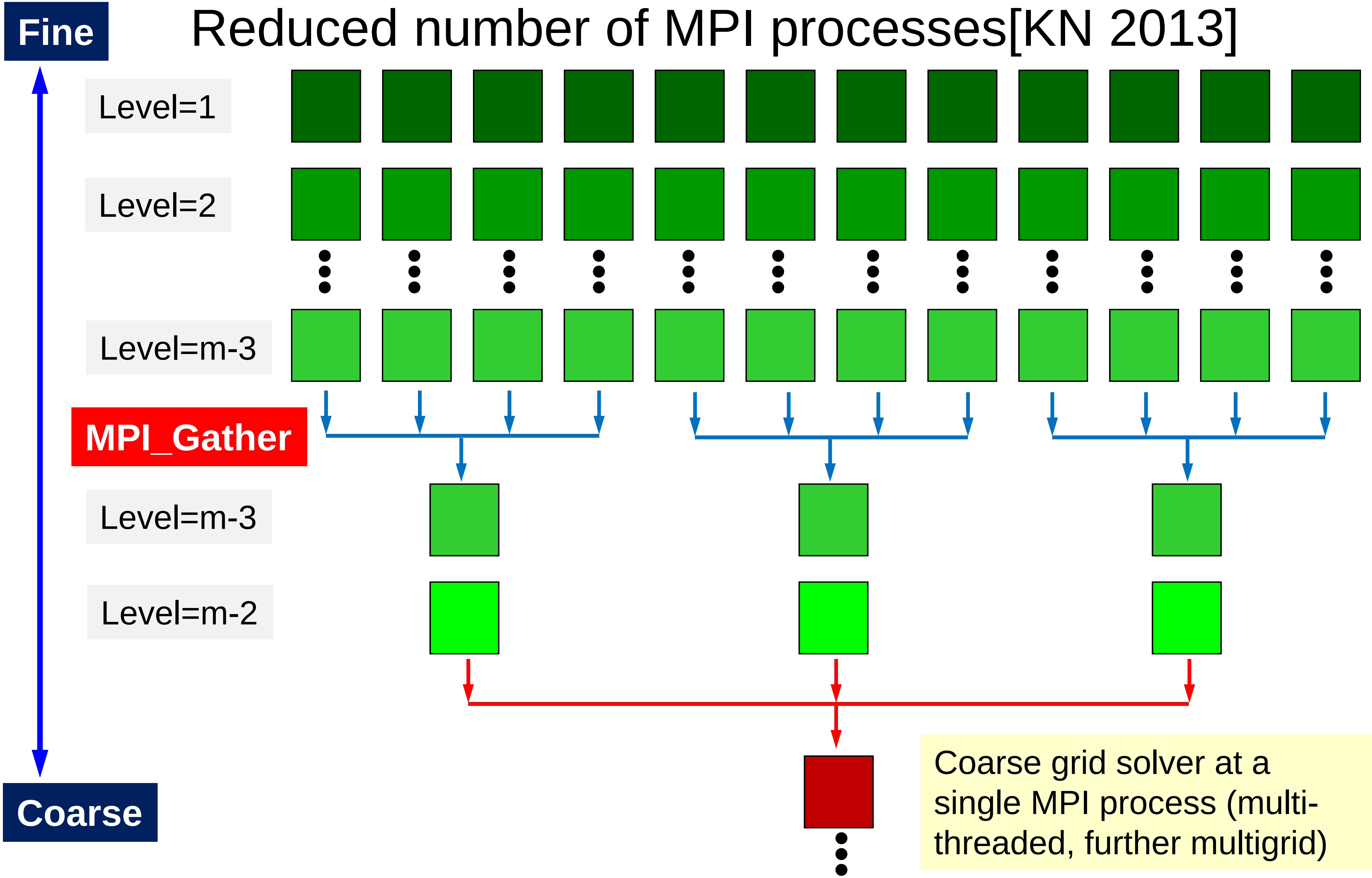
Down is good

Summary

- ELL format is effective !
- "Coarse Grid Aggregation (CGA)" is effective for stabilization of convergence at $O(10^4)$ cores for MGCG
  - HB 8x2 is the best at 4,096 nodes
- Hierarchical CGA (*h*CGA) is also effective at 4,096 nodes
- Future/On-Going Works and Open Problems
  - Algorithms
    - CA-Multigrid (for coarser levels), CA-SPAI
  - Strategy for Automatic Selection
    - optimum switching level, number of processes for *h*CGA, optimum color #
  - More Flexible ELL for Unstructured Grids
  - Optimized MPI (co-design)
    - e.g. MPI on Fujitsu FX10 utilizing RDMA with persistent communications
  - Optimum number of colors
    - strongly depends on thread #, H/W etc ...

Hierarchical CGA: Comm. Reducing MG

Reduced number of MPI processes[KN 2013]



Reducing Processes: 128->8 proc's

- MPI_Comm_split + MPI_Gather (Current)
  - (0-15):0, (16-31):16, (32-47):32 ...
  - 0, 16, 32, 48, 64, 80, 96
  - Total Send_Recv: 2.98 sec., Coarse Part: 0.25 sec.
  - MPI_Gather/Allgather: 0.046, 0.055
- MPI_Isend + MPI_Irecv (failed)
  - (0-15):0, (16-31):1, (32-47):2 ...
  - 0, 1, 2, 3, 4, 5, 6, 7
- MPI_Put + MPI_Get (not yet tried)
  - (0-15):0, (16-31):1, (32-47):2 ...
  - 0, 1, 2, 3, 4, 5, 6, 7

Send_Recv

```
!C
!C-- SEND
      do neib= 1, NEIBPETOT
        II= (LEVEL-1)*NEIBPETOT
        istart= STACK_EXPORT(II+neib-1)
        inum  = STACK_EXPORT(II+neib  ) - istart
!$omp parallel do
        do k= istart+1, istart+inum
          WS(k-NEO)= X(NOD_EXPORT(k))
        enddo

        call MPI_ISEND (WS(istart+1-NEO), inum, MPI_DOUBLE_PRECISION,    &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        II= (LEVEL-1)*NEIBPETOT
        inum  = STACK_IMPORT(II+neib) - STACK_IMPORT(II+neib-1)
        istart= NOD_IMPORT(STACK_IMPORT(II+neib-1)+1)

        call MPI_IRECV (X(istart), inum, MPI_DOUBLE_PRECISION,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (NEIBPETOT, req2, sta2, ierr)
      call MPI_WAITALL (NEIBPETOT, req1, sta1, ierr)
```

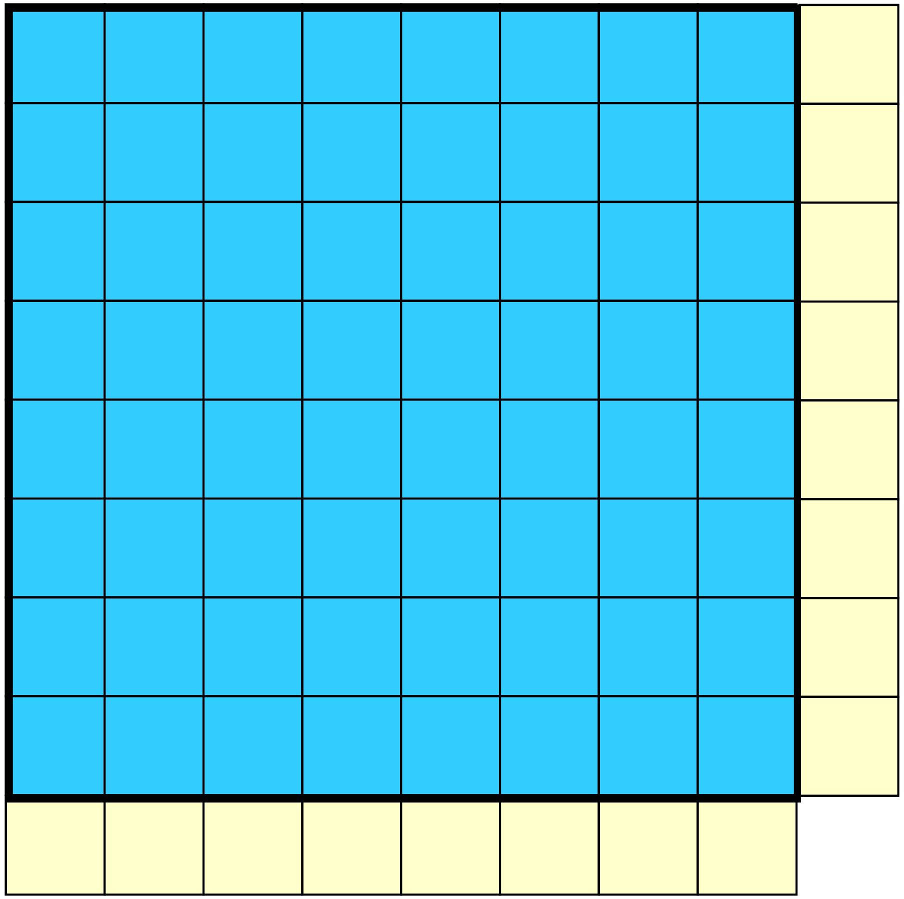
Persistent Comm. (畑中さん)（1/2）

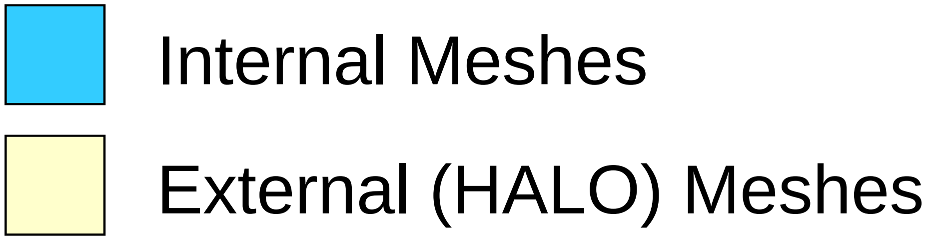
```
        allocate (REQS1(NEIBPEtot*levMGtot*2), REQS2(NEIBPEtot*levMGtot*2))

!!! REQS1 for Xmg !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        do JLEV=1, levMGtot
          JII=(JLEV-1)*NEIBPEtot
! Sender
          JNEO=EXPORT_index(JII)
          do JNEIB=1,NEIBPEtot
            JISTART= EXPORT_index(JII+JNEIB-1)
            JINUM  = EXPORT_index(JII+JNEIB)-JISTART
            call MPI_SEND_INIT(                                         &
     &         WS(JISTART+1-JNEO),JINUM,MPI_DOUBLE_PRECISION,           &
     &         NEIBPE(JNEIB),JLEV,MPI_COMM_WORLD,                       &
     &         REQS1(JNEIB+(JII*2)),JIERR)
          enddo
! Receiver
          do JNEIB=1,NEIBPEtot
            JISTART=IMPORT_index(JII+JNEIB-1)
            JINUM  = IMPORT_index(JII+JNEIB)-JISTART
            JJSTART= IMPORT_item(JISTART+1)
            call MPI_RECV_INIT(                                         &
     &         Xmg(JJSTART),JINUM,MPI_DOUBLE_PRECISION,                 &
     &         NEIBPE(JNEIB),JLEV,MPI_COMM_WORLD,                       &
     &         REQS1(JNEIB+NEIBPEtot+(JII*2)),JIERR)
          enddo
        enddo

!!! REQS2 for Rmg !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Persistent Comm. (畑中さん)（2/2）

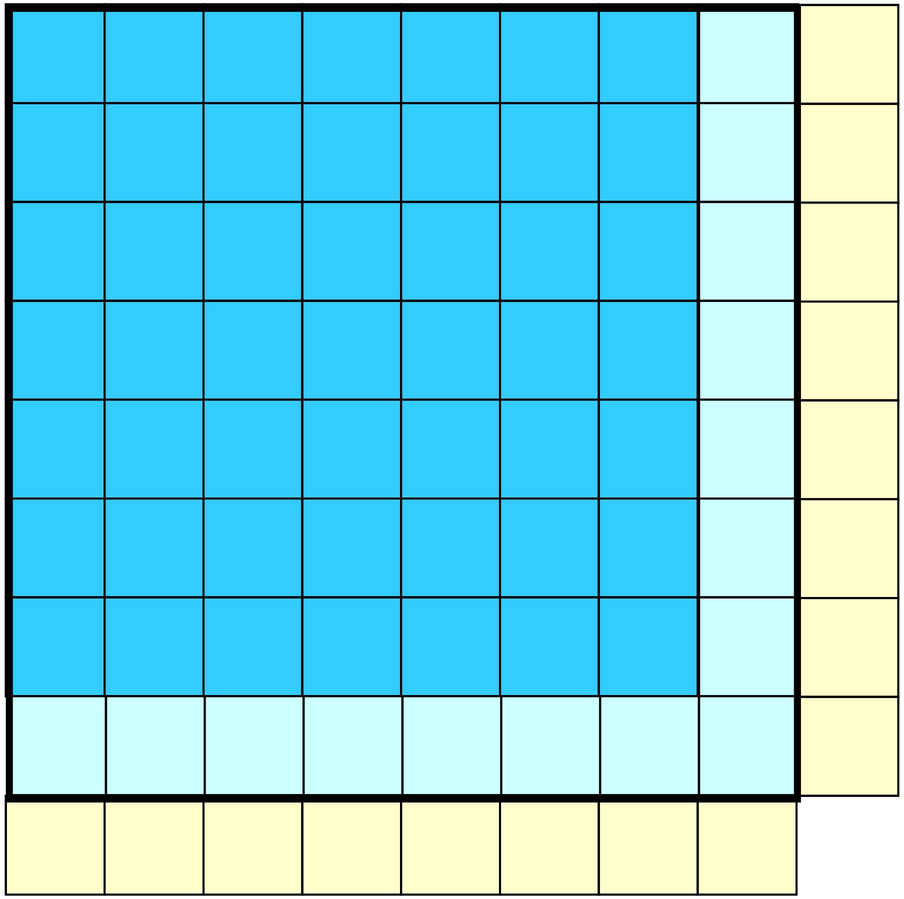
```
!C
!C-- SEND & RECV

        KII= (LEVEL-1)*NEIBPETOT
        do kneib=1, NEIBPETOT
          kistart= STACK_EXPORT(KII+kneib-1)
          kinum  = STACK_EXPORT(KII+kneib  ) - kistart
!$omp parallel do
          do kk= kistart+1, kistart+kinum
            WS(kk-NEO)=X(NOD_EXPORT(kk))
          enddo
        enddo

        call MPI_STARTALL(NEIBPETOT*2, REQS(KII*2+1), ierr)
        if (ierr .ne. MPI_SUCCESS) then
          print "(A3,I3,I5)", "#s", my_rank, ierr
          call MPI_ABORT(MPI_COMM_WORLD, ierr, ierr2)
        endif

        call MPI_WAITALL(NEIBPETOT*2, REQS(KII*2+1),                    &
     &                   MPI_STATUSES_IGNORE, ierr)

        if (ierr .ne. MPI_SUCCESS) then
          print "(A3,I3,I5)", "#w", my_rank, ierr
          call MPI_ABORT(MPI_COMM_WORLD, ierr, ierr2)
        endif
```

Comm.-Comp. Overlapping





Comm.-Comp. Overlapping



Internal Meshes

External (HALO) Meshes

Internal Meshes on Boundary's

Mat-Vec operations

- Overlapping of computations of internal meshes, and importing external meshes.
- Then computation of international meshes on boundary's
- Difficult for IC/ILU on Hybrid

Comm./Comp. Overlapping (1/3)

```
!C
!C-- SEND & RECV

        KII= (LEVEL-1)*NEIBPETOT
        do kneib=1, NEIBPETOT
          kistart= STACK_EXPORT(KII+kneib-1)
          kinum  = STACK_EXPORT(KII+kneib  ) - kistart
!$omp parallel do
          do kk= kistart+1, kistart+kinum
            WS(kk-NEO)=X(NOD_EXPORT(kk))
          enddo
        enddo

        call MPI_Isend
        call MPI_Irecv

        [Computations for Internal Nodes]

        call MPI_WAITALL for (Send+Recv)

        [Computations for Boundary Nodes]
```

Comm./Comp. Overlapping (2/3)

```
!C
!C-- SEND & RECV

        KII= (LEVEL-1)*NEIBPETOT
        do kneib=1, NEIBPETOT
          kistart= STACK_EXPORT(KII+kneib-1)
          kinum  = STACK_EXPORT(KII+kneib  ) - kistart
!$omp parallel do
          do kk= kistart+1, kistart+kinum
            WS(kk-NEO)=X(NOD_EXPORT(kk))
          enddo
        enddo

        call MPI_Isend
        call MPI_Irecv

        [Computations for Internal Nodes]

        call MPI_Waitall for Recv

        [Computations for Boundary Nodes]

        call MPI_Waitall for Send
```

Comm./Comp. Overlapping (3/3)

```
!C
!C-- SEND & RECV

        KII= (LEVEL-1)*NEIBPETOT
        do kneib=1, NEIBPETOT
          kistart= STACK_EXPORT(KII+kneib-1)
          kinum  = STACK_EXPORT(KII+kneib  ) - kistart
!$omp parallel do
          do kk= kistart+1, kistart+kinum
            WS(kk-NEO)=X(NOD_EXPORT(kk))
          enddo
        enddo

        call MPI_STARTALL(NEIBPETOT*2, REQS(KII*2+1), ierr)
        if (ierr .ne. MPI_SUCCESS) then
          print "(A3,I3,I5)", "#s", my_rank, ierr
          call MPI_ABORT(MPI_COMM_WORLD, ierr, ierr2)
        endif

        [Computations for Internal Nodes]

        call MPI_WAITALL(NEIBPETOT*2, REQS(KII*2+1),                    &
     &                   MPI_STATUSES_IGNORE, ierr)

        [Computations for Boundary Nodes]
```

Discussions

- Any Tips for MPI_Put/Get ?
- Comp./Comm. Overlapping
- Persistent Communications
- Network Topology of FX10/K